Санкт-Петербургский Государственный Университет
Фундаментальная информатика и информационные технологии

Дайнеко Роман Юрьевич

Реализация файловой системы с тонкой настройкой прав пользователей

Бакалаврская работа

Научный руководитель:
к. ф.-м. н., ассистент Ловягин Н. Ю.

Рецензент:
ассистент Чернышев Г. А.

Допущена к защите:
д. ф.-м. н., профессор Новиков Б. А.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY
Fundamental Computer Science and Information Technology

Dayneko Roman Yurievich

The implementation of a file system with fine tuning of user permissions

Bachelor's Thesis

Scientific supervisor:
Assistant Lovyagin N.Y.

Reviewer:
Assistant Chernishev G.A.

Admitted for defence:
professor Novikov B.A.

Saint-Petersburg
2016

Оглавление

Введение

Как известно, *GNU/Linux* [10] и другие *UNIX*-подобные операционные системы [4] являются многопользовательскими. Проблема организации разграничения доступа к файлам и каталогам является одним из существенных вопросов, которые должна решать операционная система. Механизмы разграничения доступа, разработанные для системы *UNIX* в 70-х годах, очень просты, но они оказались настолько эффективными, что просуществовали уже более 30 лет и по сей день успешно выполняют большинство стоящих перед ними задач. Однако порой возникают потребности, которые эти механизмы удовлетворить не в состоянии.

Сейчас права доступа к файлу регулируются 16-ю битами, хранящимися в индексном дескрипторе [14]. Первые 4 бита из которых представляют информацию о типе файла, следующие 3 бита задают особые свойства исполняемых файлов, а последние 9 кодируют информацию о правах доступа. Эти 9 бит содержат в себе 3 группы по 3 бита, предназначенные для кодирования прав доступа для владельца файла, группы-владельца файла и остальных. Сами 3 бита можно символьно представить как rwx, где r отвечает за «право на чтение», w — «право на запись», а x — «право на исполнение». Таким образом задается порядок установки прав, не позволяющий, к примеру, установить конкретному пользователю индивидуальные права доступа и, тем более, разрешить или запретить ему отдельные действия, например, разрешить ему добавлять файлы в директорию, но не удалять уже имеющиеся (возможность добавления и удаления регулируются одним битом w, т.е. либо разрешены оба, либо оба запрещены).

В данной работе решается задача реализации файловой системы, позволяющей более точно настраивать права доступа.

1. Постановка задачи

В данной главе приводится обзор существующих механизмов разграничения прав в $UNIX$-подобных системах, формулируются требования для реализуемой файловой системы, и производится постановка задач для ее достижения.

1.1. Права доступа к файлам в UNIX-подобных системах

В основе механизмов разграничения доступа лежат идентификаторы пользователей и групп пользователей. В *Linux* каждый пользователь имеет уникальный идентификатор, с которым он входит в систему. Кроме того, в системе создается некоторое число групп пользователей, причем каждый пользователь может быть включен в одну или несколько групп. Создает и удаляет группы суперпользователь, он же может изменять состав участников той или иной группы. Члены разных групп могут иметь разные права по доступу к файлам, например, группа администраторов может иметь больше прав, чем группа программистов.

В индексном дескрипторе [14] файлов записан идентификатор владельца файла и группы, которая имеет права на этот файл. Эти идентификаторы записываются в индексный дескриптор при создании, исходя из идентификаторов пользователя и группы процесса, создающего файл.

Права доступа в UNIX-системах хранятся в индексном дескрипторе в отдельной 2-х байтовой структуре вместе с информацией о типе файла и особых свойствах исполнения. На саму информацию о правах отведено 9 бит. Эти 9 бит разделяются на 3 группы по три бита. Первые 3 бита отвечают за права владельца, следующие 3 бита — права группы, последние 3 бита определяют права всех остальных пользователей (т. е. всех пользователей, за исключением владельца файла и группы файла). При этом, если соответствующий бит имеет значение 1, то право предоставляется, а если он равен 0, то право не предоставляется. В

символьной форме записи прав единица заменяется соответствующим символом (r, w или x), а 0 представляется прочерком.

Право на чтение (r) файла означает, что пользователь может просматривать содержимое файла с помощью различных команд просмотра. Но, отредактировав содержимое файла в текстовом редакторе, пользователь не сможет сохранить изменения в файле на диске, если не имеет права на запись (w) в этот файл. Право на выполнение (x) означает, что пользователь может загрузить файл в память и попытаться запустить его на выполнение как исполняемую программу. Конечно, если в действительности файл не является программой (или скриптом *shell*), то запустить этот файл на выполнение не удастся, но, с другой стороны, даже если файл действительно является программой, но право на выполнение для него не установлено, то он тоже не запустится.

Естественно, что по отношению к директориям трактовка понятий «право на чтение», «право на запись» и «право на выполнение» отличны. Право на чтение по отношению к директориям — это право на просмотр его содержимого. Право на запись — право на создание и удаление файлов в этой директории. Право на выполнение в данном случае означает право переходить в эту директорию. Если вы, как владелец, хотите предоставить доступ другим пользователям на просмотр какого-то файла в своей директории, вы должны дать им право доступа в каталог, т. е. дать им «право на выполнение директории». Более того, надо дать пользователю право на выполнение для всех директорий, стоящих в дереве выше данной директории. Поэтому для всех директорий по умолчанию устанавливается право на выполнение как для владельца и группы, так и для всех остальных пользователей. И если вы хотите закрыть доступ в каталог, то лишите всех пользователей права входить в этот каталог.

Алгоритм проверки прав пользователя при обращении к файлу можно описать следующим образом. Сперва проверяется совпадение идентификатора пользователя с идентификатором владельца файла. Если эти идентификаторы совпадают, то проверяется наличие у владелеца соответствующего права доступа: на чтение, на запись или на выпол-

нение. Если право такое есть, то соответствующая операция разрешается. Если же нужного права владелец не имеет, то проверка прав, предоставляемых через группу или через группу атрибутов доступа для остальных пользователей, уже не выполняется, а пользователю выдается сообщение о невозможности выполнения затребованного действия («*Permission denied*»).

Если идентификатор пользователя, обращающегося к файлу, не совпадает с идентификатором владельца, то система проверяет равенство идентификатора группы процесса пользователя и иденификатора группы, которая сопоставлена данному файлу (далее будем просто называть ее группой файла). Если эти идентификаторы совпадают, то для определения возможности доступа к файлу используются атрибуты, относящиеся к группе, а на атрибуты для владельца и всех остальных пользователей внимания не обращается. Если же пользователь не является владельцем файла и не входит в группу файла, то его права определяются атрибутами для остальных пользователей.

1.2. Требования к реализуемой файловой системе

С точки зрения администрирования, описанная в предыдущем пункте правовая модель навязывает достаточно жесткий порядок установки прав, что приводит к ряду неудобств.

Деление множества операций с файлами на «право на чтение», «право на запись» и «право на выполнение» делает невозможным создание тонких правовых комбинаций, что само по себе уже доставляет некоторые неудобства. К примеру, в рамках модели rwx невозможно разрешить создавать файлы в каталоге, и, при этом, запретить удалять и переименовывать существующие.

Деление множества пользователей, имеющих доступ к файлу, на «владельца», «группу файла» и «остальных» не позволяет задавать индивидуальные права для конкретного пользователя. Чтобы дать какому-то конкретному пользователю, не являющемуся владельцем, право на редактирование файла, при это не давая это право «остальным», при-

дется включить его в группу файла, имеющую это право (в противном случае это просто не представляется возможным), тем самым пользователь получит доступ ко всем файлам, к которым доступ имеет эта группа, а это, в свою очередь, при неблагоприятном исходе, может привести к довольно плачевным последствиям.

Возникает потребность в реализации файловой системы, предоставляющей инструменты для регулирования прав доступа к файлам таким образом, чтобы обеспечивалась возможность создания тонких и более индивидуальных правовых комбинаций.

1.3. Обзор существующих решений

Списки контроля доступа ACL (*Access Control Lists*) [13] позволяют установить права доступа к файлам не только для владельца и группы, но и индивидуально для любого другого пользователя или группы, без каких-либо ограничений по количеству устанавливаемых пользователей/групп. В типичных ACL каждая запись определяет субъект воздействия и операцию.

Файловая система $NTFS$ [5] семейства операционных систем $Windows$ на ядре NT [12] изначально поддерживает более гибкий механизм разграничения доступа к файлу [6]. В $GNU/Linux$ эта технология пришла относительно недавно (в ядре поддержка ACL появилась с версии 2.5), и сейчас она реализована для основных файловых систем.

Работа $POSIX$ [7] ACL базируется на использовании расширенных аттрибутов для хранения данных о правах пользователей и групп на файлы. Стоит отметить, что включение расширенных атрибутов требует перекомпиляции ядра. Эти атрибуты представляют собой список произвольных пар (имя, значение), которые привязаны к определенному индексному дескриптору файла. Имя в данных парах есть идентификатор процесса (UID или GID в терминах $POSIX$), а значение — битовая маска, символьно обозначаемая «rwx».

Существуют и проекты, реализующие централизованное хранение списков контроля доступа (т.е. списки хранятся не в расширенных ат-

рибутов каждого файла, а в отдельном файле), что в некоторых случаях упрощает администрирование. К примеру, проект *Trustees* [9] позволяет дать доступ пользователю или группе пользователей к дереву каталогов, а не единственному файлу, что, надо отметить, является отходом от стандарта *POSIX ACL*, однако сильно уменьшает размер таблицы. Стоит также отметить, что *Trustees* может быть как скомпилирован вместе с ядром, так и установлен как модуль, что значительно удобнее.

Тем не менее, ни одна из технологий, реализующих *ACL* для *GNU/Linux*, не расширяет модель *rwx*, что говорит об актуальности преследуемой цели.

1.4. Постановка задачи

Целью данной работы является получение комплекса программного обеспечения — файловой системы, сочетающей в себе удобство существующих подходов и, при этом, дающей возможность более тонкой настройки прав пользователей.

Для достижения цели были поставлены следующие задачи:

- разработка правового набора, обеспечивающего возможность создания тонких правовых комбинаций;
- разработка алгоритма проверки прав новго правового набора;
- разработка принципов трансляции между существующей моделью и новой;
- реализация комплекса ПО.

1.5. Резюме

В данной главе были обозначены цель и задачи для ее достижения, основанные на недостатках существующих механизмов разграничения прав в *UNIX*-подобных системах.

2. Реализация

В данной главе приводится описание решений задач, поставленных в предыдущей главе, а также приводится описание инструментов разработки.

2.1. Выбор инструментов разработки

Для написания собственных файловых систем в ядре *Linux* с версии 2.6.14 [8] присутствует модуль *FUSE* (*Filesystem in Userspace*) [11]. Он позволяет создавать собственные файловые системы, не переписывая код ядра. Это достигается засчет запуска кода файловой системы в пространстве пользователя, в то время как модуль *FUSE* только предоставляет мост для интерфейсов ядра. Файловые системы на *FUSE* — портируемы, причем реально файлы могут храниться на почти любой дисковой файловой системе, быть доступны по сети или же вовсе быть виртуальными. Стоит также отметить, что *FUSE* обладает доказанной стабильностью.

Также комплексу ПО необходима база данных (по причинам, описанным в пункте 2.4). Из систем управления базами данных была выбрана *SQLite* [1] ввиду того, что движок *SQLite* не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций библиотеки *SQLite*. Также *SQLite* хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле. Такой подход уменьшает накладные расходы, время отклика и упрощает программу, а также не требует установки стороннего программного обеспечения.

Модуль *FUSE* написан на языке *C*, но существуют привязки и к другим языкам; тоже самое можно сказать и о библиотеке *SQLite*, однако ввиду первичной интеграции, для разработки комплекса программного обеспечения язык *C* был выбран как основной.

Для поиска и устранения утечек и ошибок при обращении к памяти использовалась утилита $Valgrind$ [2].

2.2. Расширение правовго набора

Для написания файловой системы, используя $FUSE$, необходимо, если не вдаваться в детали реализации, переопределить файловые операции. В зависимости от версии, $FUSE$ предоставляет возможность переопределения от тридцати до пятидясети одной файловых операций, ни одна из которых не является абсолютно необходимой, но многие нужны для корректной работы файловой системы.

Создание правовго набора, включающего в себя права на выполнение отдельно каждой из этих операций, влечет за собой возникновение парадоскальных ситуаций, вроде возможности открыть файл, но невозможности его закрыть. Поэтому из этого множества был выбран набор операций, реализующий основной функционал файловой системы, не допускающий возникновение парадоксов (это обеспечивается независимостью этих прав, т.е. выполнение одного не требует последующего выполнения другого), и для него создан правовой набор, позволяющий создавать достаточно тонкие правовые комбинации. Название каждого права порождается названием соответствующей операции ядра.

Набор прав для файлов выглядит следующим образом:

- $getattr$ — право на получение атрибутов файла;
- $fgetattr$ — право на получение атрибутов открытого файла;
- $readlink$ — право чтения объекта символической ссылки;
- $read$ — право на чтение содержимого файла;
- $write$ — право на запись в файл;
- $unlink$ — право удаления файла;
- $link$ — право создания жесткой ссылки на файл;

- *symlink* — право создания символической ссылки;
- *chmod* — право изменения прав доступа к файлу;
- *chown* — право изменения владельца и группы файла;
- *utime* — право изменения времени доступа и модификации файла;
- *rename* — право изменения пути к файлу внутри файловой системы;
- *execute* — право на выполнение.

Стоит отметить, что права *read*, *write* и *execute* данного набора отвечают конкретно за операции *read*, *write* и *execute*, а не сочетают в себе права на множество операций, как в модели *rwx*.

Директориям для обеспечения возможности регулирования возможностей действий с файлами внутри директории необходим иной правовой набор:

- *getattr* — право на получение атрибутов директории;
- *fgetattr* — право на получение атрибутов открытой директории;
- *readlink* — право чтения объекта символической ссылки, если последняя в директории;
- *read* — право на чтение содержимого файлов в директории;
- *write* — право на запись в файлы директории;
- *readdir* — право на получение списка файлов в директории;
- *opendir* — право перехода в директорию;
- *mknode* — право создания файла в директории;
- *mkdir* — право создания поддиректорий в директории;
- *unlink* — право удаления файла в директории;

- *rmdir* — право удаления директории и всего ее содержимоего;
- *link* — право создания жесткой ссылки в директории;
- *symlink* — право создания символической ссылки в директории;
- *chmod* — право изменения прав доступа к директории;
- *chmoddir* — право изменения прав доступа к файлам в директории;
- *chown* — право изменения владельца и группы директории;
- *chowndir* — право изменения владельца и группы файлов в директории;
- *utime* — право изменения времени доступа и модификации директории;
- *utimedir* — право изменения времени доступа и модификации файлов в директории;
- *rename* — право изменения пути к директори внутри файловой системы;
- *renamedir* — право изменения пути к файлам в директории внутри файловой системы;
- *execute* — право на выполнение файлов в директории.

Права на опреации, не вошедшие в правовую модель, либо не несут какой-либо смысловой нагрузки, как, например, право на выполнение операции *access* (проверка наличия прав на какое-либо действие), либо могут быть целиком определены через другие права, например, на право выполнение опреации *open* (операция открытия файла) может быть целиком определено через права *read* и *write*, в зависимости от намерений на открытие, либо же соответствующая им операция является следствием какой-либо другой операции. Примером этому может

послужить операция *release*, которая вызывается каждый раз после выполнения *open*.

Помимо расширения модели *rwx*, файловая система должна иметь возможность настройки индивидуальных прав для пользователей и групп. Это означает, что система будет иметь пять типов правовых наборов. А именно: права владельца файла, права группы файла, индивидуальные права пользователя на файл, индивидуальные права группы на файл и права для «остальных».

2.3. Алгоритм проверки прав

Алгоритм сопоставления правового набора для процесса при обращении к файлу в новой правовой модели построен следующим образом:

1. Сначала проверяется наличие индивидуальных прав на действие с файлом по идентификатору пользователя (UID) процесса, если они существуют, то они и будут использоваться для определения доступа к файлу.

2. Если таковых прав не оказалось, проверяется наличие индивидуальных прав для группового идентификатора (GID) процесса, и при их существовании, определение возможности действия предоставляется им.

3. Иначе проверяются на соответствие идентификатор пользователя и идентификатор владельца файла, и при их совпадении, возможность доступа определяется правами владельца.

4. Если эти идентификаторы не совпали, то возможность доступа определяется правами для сопоставленной файлу группы, если групповой идентификатор процесса совпадает с идентификатором группы файла.

5. В противном случае возможность доступа определяется правами для «остальных».

6. Полученный правовой набор конъюктируется с правовым набором на наддиректорию, полученным таким же образом.

Стоит отметить, что индивидуальные права на файл для пользователя или группы не обязательно существуют, тогда как права владельца, группы файла и «остальных» существуют всегда, однако индивидуальные права имеют больший приоритет нежели стандартные.

2.4. Трансляция стандартного и нового правовых наборов

Права стандартного правового набора *read*, *write* и *execute* являются объединением прав набора предложенного в пункте 2.2. Это означает, что можно легко без потерь смысла представить набор прав модели *rwx* через права расширенной модели. Для файлов это можно сделать следующим образом:

- $r \to (read, getattr, fgetattr, readlink)$;
- $w \to (write, unlink, rename, link, symlink, utime, chmod, chown)$;
- $x \to (execute)$.

Для директорий:

- $r \to (read, getattr, readdir, fgetattr, readlink)$;
- $w \to (write, unlink, rename, renamedir, link, mknod, mkdir, rmdir, symlink, utime, utimedir, chmod, chmoddir, chown, chowndir)$;
- $x \to (execute, opendir)$.

Словами эти правила можно интерпритировать как «наличие права a означает наличие множества прав (b) в новой правовой модели». Очевидно, что правовой набор модели *rwx*, являющийся конкатенацией прав r, w и x, может быть транслирован в новую правовую модель, исходя из этих правил.

Трансляция в обратную сторону без потерь невозможна. Было решено сопоставлять соответствующий бит стандартной правовой модели расширенному правовому набору, если последний имеет хотя бы одно право (соответствующий праву бит равен 1), за которое отвечает этот бит. Другими словами, правила трансляции расширенной правовой модели в стандартную для файлов выглядят следующим образом:

- $read \vee getattr \vee fgetattr \vee readlink \rightarrow r$;
- $write \vee unlink \vee rename \vee link \vee symlink \vee utime \rightarrow w$;
- $execute \rightarrow x$.

Для директорий:

- $read \vee getattr \vee readdir \vee fgetattr \vee readlink \rightarrow r$;
- $write \vee unlink \vee rename \vee renamedir \vee link \vee mknod \vee mkdir \vee rmdir \vee symlink \vee utime \vee utimedir \vee chmod \vee chmoddir \vee chown \vee chowndir \rightarrow w$;
- $execute \vee opendir \rightarrow x$.

Сама трансляция необходима для корректной работы операций *chmod*, *getattr* и *fgetattr*. В первом случае (*chmod*) производится трансляция из стандартной правовой модели в новую для выставления прав стандартным образом. Во втором и третьем случае (*getattr* и *fgetattr*) производится трансляция в обратную сторону.

2.5. Принципиальное решение

Разрабатываемый комплекс программного обеспечения было решено составить из двух компонент: монтируемого драйвера файловой системы и программы администрирования файловой сисетемы. Драйвер получил название *FTFS* (*Fine Tunning FileSystem*), а программа администрирования — *FTPC* (*Fine Tunning Permission Control*).

Запросы на файлы процессов пользователя выполняются следующим образом: сначала запрос поступает в ядро, которое делегарует его $FUSE$-модулю, который, в свою очередь, делегирует его непосредственно виртуальной файловой системе.

Для того чтобы обеспечить пользователям нашей файловой системы доступ по расширенной правовой модели, было решено использовать дисковую файловую систему для физического хранения файлов и отображать их пользователю, используя расширенную правовую модель. Другими словами: $FTFS$ будет иметь полный доступ к файлам, которые будут храниться с помощью дисковой файловой системы и предоставлять доступ пользователям по расширенным правам. Таким образом, задача нашей виртуальной системы состоит в том, чтобы обработать запрос и нужным образом вызвать функции ядра, предназначенные для работы с дисковой файловой системой.

Драйвер файловой системы при монтировании будет получать на вход две директории: точку монтирования системы (далее *mountdir*), и директорию, в которой будут физически храниться файлы (далее *rootdir*). Пользователи имеют доступ к *mountdir*, но не имеют доступа к *rootdir*, а драйвер имеет полный доступ к *rootdir* и отображает файлы из *rootdir* в *mountdir* и обратно, используя относительные пути файлов. Такой подход необходим ввиду того, что драйвер файловой системы должен иметь полный доступ ко всем файлам независимо от настроек прав на них.

Отдельная программа, предназначенная для администрирования, нужна потому, что невозможно, используя $FUSE$, переопределить операцию *chmod* (операция изменения прав доступа) так, чтобы принимался набор аргументов, позволяющий изменять права отдельных пользователей или групп.

В качестве способа хранения информации о доступе была выбрана база данных, она получила название $FTDB$ ($Fine\ Tunning\ DataBase$). Выбор обусловлен тем, что такой подход обеспечивает максимальную портиемость и не требует дополнительных взаимодействий с ядром, т.к. альтернативный способ хранения (расширенные атрибуты) поддержи-

вается не всеми файловыми системами, на которых будет храниться *rootdir* со всем ее содержимым, и в некоторых случаях требует перекомпиляции ядра для включения.

Блок-схема взаимодействия компонент комплекса, базы данных, ядра, *FUSE*-модуля, дисковой фаловой системы и пользователя представлена на Рис. 1.

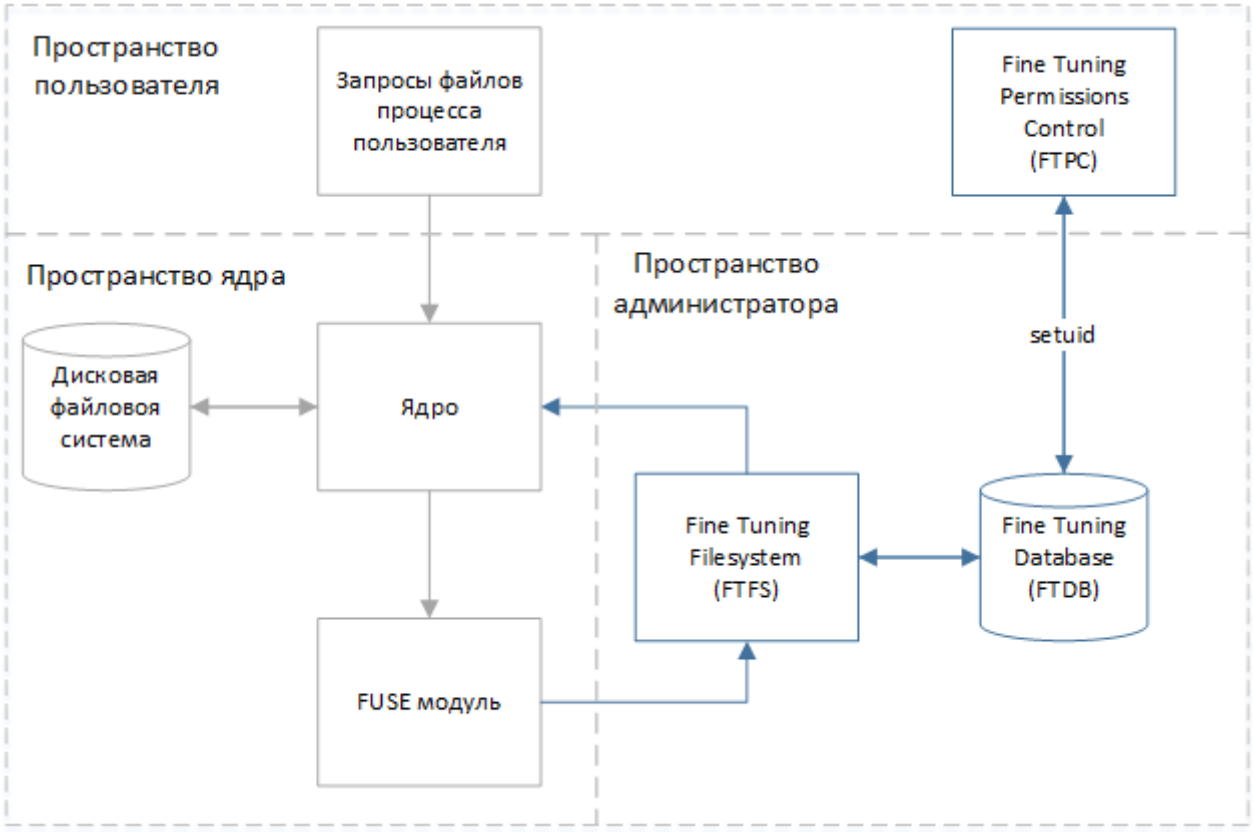

Рис. 1: Блок-схема работы разрабатываемого комплекса ПО

Как уже говорилось, сперва запрос файлов пользовательского процесса поступает в ядро, которое делегирует его *FUSE* модулю, который, в свою очередь, делегирует его *FTFS*. Драйвер, используя *SQLite API*, сверяет наличие прав на соответствующее действие с данными *FTDB* и вызывает соответствующие функции ядра, предназначенные для работы с дисковой файловой системой, в случае, если действие разрешено.

За установку и изменение прав отвечает *FTPC*, который запускается отдельным процессом в пространстве пользователя. *FTPC* прове-

ряет наличие права на изменение прав у вызвавшего его пользователя и изменяет данные $FTDB$ в случае, если такое право есть. Чтобы получить доступ к $FTDB$, находящейся в пространстве администратора, $FTPC$ использует функцию *setuid*, которая позволяет процессу выполняться от пользователя *root*.

База данных состоит из следующих компонент: таблицы конфигурации файловой системы, таблицы со списком файлов, таблицы, содержащей список индивидуальных прав пользователей и таблицы со списком индивидуальных прав для групп. Подробное описание таблиц и их полей представлено на Рис. 2.

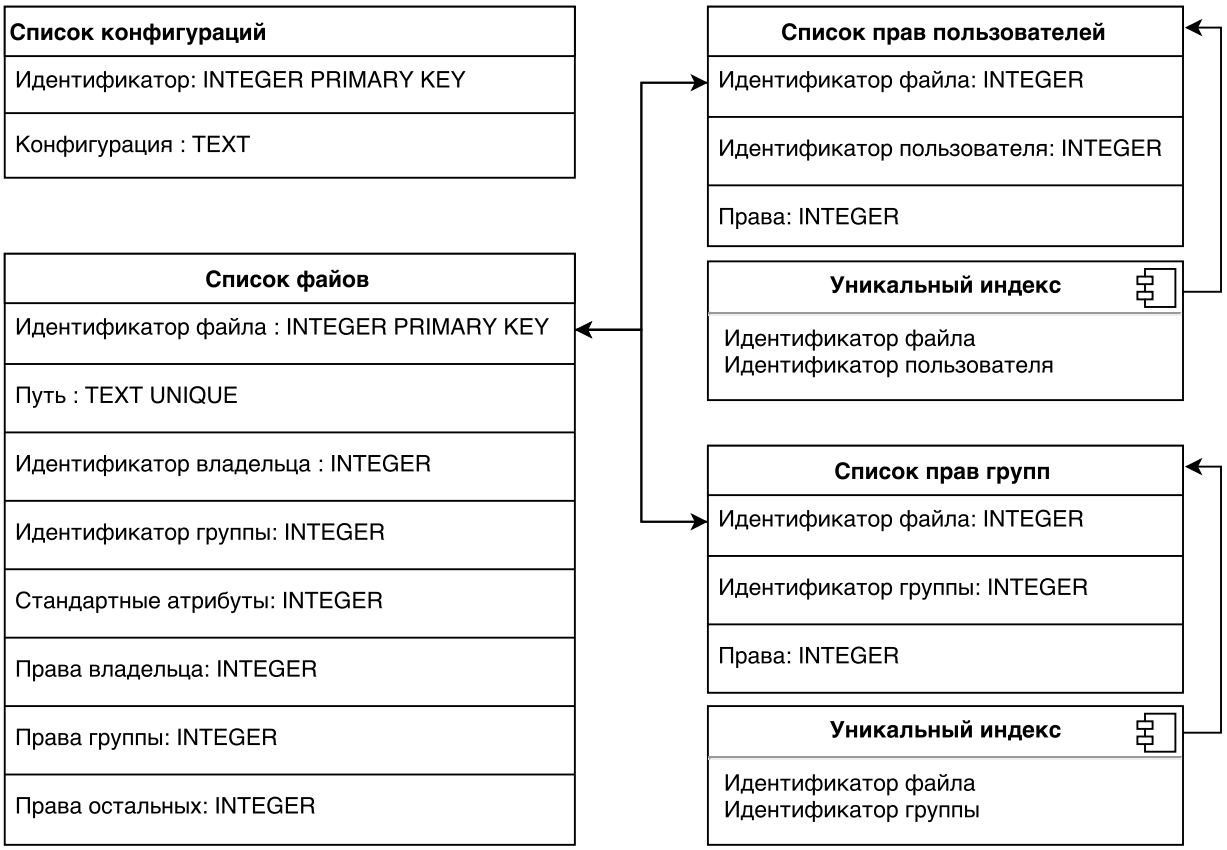

Рис. 2: Структура $FTDB$

Таблица конфигурации файловой системы необходима для работы $FTPC$. $FTPC$ в качестве первого аргумента при запуске получает путь к базе данных и извлекает из таблицы конфигурации пути к *rootdir* и *mountdir*.

Таблица, содержащая список файлов, хранит каждый файл в един-

ственном экземпляре (благодаря наложенному на поле «Путь» ограничению $UNIQUE$) и реализует связь «один ко многим» с таблицами, содержащими список прав пользователей и групп через поле «Идентификатор файла».

Уникальные индексы обеспечивают хранение записей прав для отдельно взятого файла и группы или пользователя в единственном экземпляре.

2.6. Резюме

В данной главе были разработаны новая правовая модель и алгоритм праверки прав этой модели. Также было приведено описание структуры программного комплекса предстоящего к реализации.

3. Описание комплекса программного обеспечения

В данной главе приводятся основные аспекты реализации программного комплекса.

3.1. FTFS

Чтобы написать файловую систему, используя $FUSE$-модуль, нужно переопределить файловые операции. Это происходит путем передачи при монтировании файловой системы $FUSE$-модулю структуры $fuse\_operations$, хранящей в полях ссылки на функции файловых операций. Как уже говорилось, $FUSE$-модуль не требует переопределения всех функций структуры $fuse\_operations$ (непереопределенные функции будут выполняться стандартным образом). Прототипы функций из этой структуры, переопределяемых в $FTFS$ операций, представлены на Листинге 1. Переопределяемые функции были выбраны исходя из того, что никакая из них не является конкатенацией других, и при этом вместе они обеспечивают корректную работу файловой системы. Подробно про назначение функций и принимаемых ими аргументах написано в [3].

Листинг 1: Прототипы функций операций $FTFS$

```
int (*getattr) (const char *, struct stat *);
int (*readlink) (const char *, char *, size_t);
int (*mknod) (const char *, mode_t, dev_t);
int (*mkdir) (const char *, mode_t);
int (*unlink) (const char *);
int (*rmdir) (const char *);
int (*opendir) (const char *, struct fuse_file_info
  *);
int (*readdir) (const char *, void *,
  fuse_fill_dir_t, off_t, struct fuse_file_info);
int (*symlink) (const char *, const char *);
```

```
int (*rename) (const char *, const char *);
int (*link) (const char *, const char *);
int (*chmod) (const char *, mode_t);
int (*chown) (const char *, uid_t, gid_t);
int (*ftruncate) (const char *, off_t, struct
  fuse_file_info *);
int (*truncate) (const char *, off_t);
int (*utime) (const char *, struct utimbuf *);
int (*open) (const char *, struct fuse_file_info *);
int (*read) (const char *, char *, size_t, off_t,
  struct fuse_file_info *);
int (*write) (const char *, const char *, size_t,
  off_t, struct fuse_file_info *);
int (*statfs) (const char *, struct statfs_t *);
int (*flush) (const char *, struct fuse_file_info *);
int (*release) (const char *, struct fuse_file_info
  *);
int (*fsync) (const char *, int, struct
  fuse_file_info *);
int (*fgetattr) (const char *, struct stat *, struct
  fuse_file_info *);
int (*access) (const char *, int);
```

Этими прототипами для каждой функции задается жесткий (без возможности изменения) набор аргументов, и именно этим обуславливается существование *FTPC*.

Проверка прав осуществляется в каждой из переопределенных функций (на соответствующую функции операцию) перед непосредственным вызовом функций ядра дисковой файловой системы. В случае, если права на соответствующую операцию не оказалась, функции возвращают ошибку, которая интерпритируется операционной системой как *Permission denied*. Каждую переопределенную функцию можно описать шаблоном представленным на Листинге 2.

Стоит также отметить, что *fuse_operations* не представляет возможности переопределения операции выполнения, однако возможность ее выполнения можно регулировать путем переопределения соответствующим образом операции *access*. Другими словами: операция *access* переопределена таким образом, что в ней проверяется только возможность выполнения. На остальные действия выдается разрешение, однако они могут быть запрещены в последующем вызове конкретной функции операции.

Листинг 2: Шаблон функции операции *FTFS*

```
int ft_<operation name> (const char *path, <operation
  args>)
{
    int retstat = 0;
    log_msg(<operation name>, path, <operation args>);

    int permission = <get_prms|get_dir_prms>(path,
      fuse_get_context()->uid,
      fuse_get_context()->gid) & <operation define>;

    if (!permission)
        retstat = -EACCESS;
    else
    {
        retstat = <disc filesystem operation call>;
        if (retstat)
        {
            <data base operation call>
            <another functional>
        }
    }
    return retstat;
}
```

В этом шаблоне:

- $< operation\ name >$ — имя соответствующей операции;
- $< operation\ args >$ — аргументы соответствующей операции;
- $< get\_prms|get\_dir\_prms >$ — функции, возвращающие битовую маску прав на данный файл для процесса с данным *uid* и *gid* для файла или директории (get_dir_prms вызывается в функциях ft_mkdir и ft_mknod из-за того, что файла еще не существует в базе данных и право на их выполнение регулируется правами на наддиректории) (сами эти функции будут описаны далее).;
- $< operation\ define >$ — правовой бит для данной файловой операции;
- $< disc\ filesystem\ operation\ call >$ — вызов соответствующей операции ядра с дисковой файловой системой.
- $< data\ base\ operation\ call >$ — операция изменения данных $FTDB$ (это может быть добавление, обновление или удаление записи в зависимости от операции), которая может потребоваться после вызова $< disc\ filesystem\ operation\ call >$;
- $< another\ functional >$ — дополнительная функциональность, к примеру, запись информации в буфер.

Прототипы функций $FTFS$ работы с $FTDB$ представлены на Листинге 3.

Листинг 3: Прототипы функций работы с $FTDB$

```
int get_dir_prms(const char *, const uid_t *, const
  gid_t *, const int);
int get_prms(const char *, const uid_t *, const gid_t
  *);
int add_file(const char *, const uid_t *, const gid_t
  *, const mode_t);
```

```
int update_owners(const char *, const uid_t *, const
  gid_t *);
int update_path(const char *, const char *);
int update_permissions(const char *, const mode_t);
int remove_file(const char *);
int init_db(const char *, const char *, const uid_t
  *, const gid_t *);
```

Функции *get_prms* и *get_dir_prms* реализуют алгоритм определения прав, т.е. алгоритм сопоставления набора прав для данных *uid* и *gid* процесса и указанного файла, однако не выполняют саму проверку, она, как говорилось в предыдущем пункте, выполняется непосредственно в каждой функции операций. Само сопостовление осуществляется по изложенному в предыдущей главе алгоритму. Функция *add_file* создает запись в таблице *file_list* из *FTDB* с данными файла. Функции *update_owners*, *update_path* и *update_permissions* обновляют соответствующие поля таблицы *file_list*. Эти функции вызываются в функциях *FTFS* операций *chown*, *rename* и *chmod* соответственно. Функция *remove_file* удаляет запись из таблицы *file_list*, она вызывается в функциях *Unlink* и *Rmdir*. Функция *init_db* создает саму *FTDB* и таблицы в ней. Файл *FTDB* было решено поместить в корень *rootdir*, а файлы *FTFS* в *rootdir/content* для обеспечения однозначности пренадлежности *FTDB* экземпляру *FTFS*. Функция *init_db* вызывается перед непосредственным монтированием файловой системы. Все функции, кроме *init_db*, устойчивы к ошибкам и не прерывают работу драйвера в случае их возникновения. Возникновение ошибки в *init_db* приводит к отказу в монтировании *FTFS*.

FTFS получает на вход путь к *rootdir* и *mountdir* в качестве двух последних аргументов. Перед ними могут передаваться опции монтирования для *FUSE*-драйвера и *--log* для включения логирования всех выполненных операций файловой системы.

3.2. FTPC

Для того, чтобы получить доступ к $FTDB$, $FTPC$ должен запускаться в пространстве администратора, т.е. только администратор файловой системы может настраивать права пользователей. Однако это решение сильно ограничивает количество файлов файловой системы (в том смысле, что администрирование всех файлов возлагается на одного человека). Поэтому было решено раздать $FTPC$ каждому пользователю файловой системы, если последний желает управлять правами файлов, при этом, естественно, проверяя право на изменине прав (*chmod*) для данного пользователя и файла, права которого этот пользователь хочет изменить. Однако у такого решения была проблема. $FTDB$ находится в пространстве администратора $FTFS$, а $FTPC$ теперь выполняется еще и в пространствах пользователей. Чтобы $FTPC$ имел доступ к $FTDB$, было решено использовать системный вызов *setuid*, который давал бы возможность выполняться от *root*, и тем самым получать доступ к пространству администратора $FTFS$. Для того, чтобы использовать *setuid*, $FTPC$ должен обладать особым битом исполняемых файлов.

Функциональность $FTPC$ составляют пять команд, которые передаются ему в качестве аргументов при вызове: --*help*, --*show*, --*set*, --*change* и --$configure$.

Команда --*show* выводит информацию хранящуюся в $FTDB$ на консоль, --*set* выставляет права с помощью битовой маски, --*change* добавляет или удаляет права по типу, а --$configure$ используется для переключения между различными смонтированными $FTFS$. Параметры вызовов команд представлены на Таблице 1.

Таблица 1: Команды $FTPC$

Команды	Опции	Аргументы	Описание
--h --$help$			Вывод подобной таблицы
--cfg --$configure$		Путь к $FTDB$	Переключение на работу с экземпляром $FTDB$
--set	-ow	Путь к файлу, правовая маска	Установка прав владельца по маске
	-gow	Путь к файлу, правовая маска	Установка прав группы файла по маске
	-oth	Путь к файлу, правовая маска	Установка прав для «остальных» по маске
	-u	Путь к файлу, идентификатор пользователя, правовая маска	Установка индивидуальных прав для пользователя по маске
	-g	Путь к файлу, идентификатор группы, правовая маска	Установка индивидуальных прав для группы по маске
--c --$change$	-ow	Путь к файлу, список <±Право>	Изменение прав владельца по типу
	-gow	Путь к файлу, список <±Право>	Изменение прав группы файла по типу
	-oth	Путь к файлу, список <±Право>	Изменение прав для «остальных» по типу
	-u	Путь к файлу, идентификатор пользователя, список <±Право>	Изменение индивидуальных прав для пользователя по типу

Команды	Опции	Аргументы	Описание
--*c* --*change*	-*g*	Путь к файлу, идентификатор группы, список <±Право>	Изменение индивидуальных прав для группы по типу
--*show*		Путь к файлу	Вывод информации из $FTDB$ относительно файла
		all	Вывод всей содержащейся в $FTDB$ информации
		(*null*)	Вывод всей содержащейся в $FTDB$ информации

В этой таблице <±Право> — обозначение для строк вида «-Право» и «+Право», которые в свою очередь обозначают удалить и добавить право соответственно. Само слово «Право» обозначает название конкретного права, список которых был предложен в пункте 2.2.

3.3. Резюме

В данной главе были приведены описания основной структур и функциональности драйвера и программы администрирования файловой системы.

4. Заключение

Выпускная квалификационная работа посвящена вопросу реализации файловой системы с тонкой настройкой прав пользователей. Данный вопрос актуален ввиду значительного усложнения процесса администрирования файловых систем с ростом числа ее пользователей.

В рамках данной работы были рассмотрены существующие механизмы разграничения доступа UNIX-подобных систем и выделены их недостатки, исходя из которых были разработаны новая правовая модель и алгоритм проверки прав новой правовой модели, обеспечивающие возможность более гибкой настройки прав пользователя. Используя эти наработки, были разработаны драйвер файловой системы и программа администрирования файловой системы (суммарно $\sim$2500 строк кода).

Для реализации использовался язык программирования C, $FUSE$-модуль ядра операционной системы $GNU/Linux$ и система управления базами данных $SQLite$, что обеспечивает максимальную портируемость решения.

Таким образом, задачи, поставленные для достижения цели, решены в полном объеме, цель достигнута.

Список литературы

[1] Consortium SQLite. SQLite // SQlite. — 2016. — URL: https://www.sqlite.org/about.html (online; accessed: 09.01.2016).

[2] Developers Valgrind. Valgrind // Valgrind. — 2016. — URL: http://valgrind.org (online; accessed: 10.03.2016).

[3] Fossies. fuse Documentation // Software Archive. — 2016. — URL: https://fossies.org/dox/fuse-2.9.6/structfuse__operations.html (online; accessed: 09.01.2016).

[4] Group The Open. IEEE Std 1003.1™, 2013 Edition // Single UNIX Specification. — 2013. — URL: http://pubs.opengroup.org/onlinepubs/9699919799/ (online; accessed: 09.02.2016).

[5] Inc. LSoft Technologies. NTFS // NTFS.com. — 2016. — URL: http://ntfs.com/ntfs.htm (online; accessed: 20.04.2016).

[6] Inc. LSoft Technologies. NTFS Permissions // NTFS.com. — 2016. — URL: http://ntfs.com/ntfs-permissions.htm (online; accessed: 20.04.2016).

[7] Josey Andrew. IEEE Computer Society // POSIX - Austin Joint Working Group. — 2016. — URL: http://standards.ieee.org/develop/wg/POSIX.html (online; accessed: 15.03.2016).

[8] Linux. Linux.com archive // 2.6.14. — 2005. — URL: http://archive09.linux.com/feed/47839 (online; accessed: 08.01.2016).

[9] Ruder Andrew. Trustees ACL // Trustees. — 2005. — URL: http://trustees.sourceforge.net (online; accessed: 20.04.2016).

[10] Stallman Richard. GNU // Linux and the GNU System. — 2016. — URL: http://www.gnu.org/gnu/linux-and-gnu.en.html (online; accessed: 08.01.2016).

[11] Wikipedia. Filesystem in Userspace // the free encyclopedia. — 2016. — URL: https://en.wikipedia.org/wiki/Filesystem_in_Userspace (online; accessed: 09.01.2016).

[12] Wikipedia. Windows NT // the free encyclopedia. — 2016. — URL: https://en.wikipedia.org/wiki/Windows_NT (online; accessed: 20.04.2016).

[13] user community Arch Linux. Arch Linux // Access Control Lists. — 2016. — URL: https://wiki.archlinux.org/index.php/Access_Control_Lists (online; accessed: 01.02.2016).

[14] Костромин Виктор. Linux для пользователя. — БХВ-Петербург, 2003. — С. 658.